A Graphical Editor for LOA

Rich Graphical Components Design and Integration

Olivier Beaudoux
Mickael Clavreul
ESEO Group
2012-2013
The LOA Project

LOA: Langage pour Opérations Actives (Language for Active Operations)

Projet MPIA

Objectives:
Design Graphical Components and Interactions
Integration of multiple stakeholders designs
Automated Synthesis of Rich Graphical Interfaces
MVC: Model-View-Controller

**MVC**

Model: defines objects of the domain
View: presents objects to the user
Controller: interprets interactions and synchronize model & view

Bindings: ensure consistency

**Typical Flow**

1. User interacts
2. The controller modifies the internal model
3. The model notifies its view
4. The view send feedback to the user
MVC Issues

Specification of the Graphical User Interface

Specification of View & Interactions unsupported
Specification of bindings between “views” unsupported

Programming effort

Programming View, Interactions and bindings is risky
Reusing graphical designer sketches is expensive
Integration of the model-view-controller is time-consuming
LOA Approach

Views
- Domain Objects: model of the application
- Graphical Templates: specification and parameterization of the graphical components
- Actions: set of available actions for the application
- Interactions: set of interactions captured and processed by the application

Bindings
- Mappings: specification of the integration of the views
- Mixin: create interactors that react to user's commands
- Pickings: selection of templates and objects impacted by an interaction
**LOA API**

Core: access to the core elements of LOA
UI: specification of generic widgets
Toolkits: adapters to existing toolkits (e.g. Swing, SVG)

**LOA DSL**

supports the specification of templates, interactors and data model within the same language

**LOA Application**

is automatically generated from the specifications and runs on top of the LOA API.
LOA Framework

LOA Process

- A generic process that handles both design, specification and integration phases:
- Reconciles multi-views from multiple stakeholders
- Synthesizes components to produce the graphical interface
LOA Integrator

Needs

Specification and reuse of multiple views from multiple stakeholders

User-friendly interface for the integration of the views
Specify the Domain Data

Supports packages, classes, datatypes, enumerations and the full range of relationships

Similar to the Ecore graphical editor for OO modelling
Specify the Graphical Templates

Supports the creation of the presentation classes (i.e. templates) and references to widgets if necessary
Specify the Graphical Templates

Embedded LOA language editor for parameterizing the templates
Specify the Data Bindings

Supports the creation of class-mappings between objects of the domain and templates, or between objects of the domain and widgets.

Class-mappings specify how instances of templates or widgets are created.
Specify the Data Bindings

Embedded LOA language editor for the specification of bindings
Specify the Interactors

Designers can import predefined interactors
Specify the Interactors

Designers can import predefined interactors
A set of instances inferred from an existing graphical user interface can be inferred.
Specify the Interactors

Designers can import predefined interactors

A set of instances inferred from an existing graphical user interface can be inferred.

InstanceMappings define bindings between objects, templates, widgets or interactors instances to specify the behavior of interactors.
Specify the Interactors

Embedded LOA language editor for the specification of bindings
Running a LOA application

Synthesis of the graphical interface

Inputs

- LOA Model created with the graphical designer
- a folder and a package for the generated sources
- Filenames for the generation of the four views (i.e. domain objects, templates, interactors)
- the LOA application filename
- Data from the introspection phase if any (Introspection is the analysis of an existing graphical interface to detect instances of existing graphical components)

Output

- a complete application integrating the four views that runs on a given execution platform (e.g. Swing, SVG)
Running the LOA application

the LOA framework integrates the four views into a fully synchronized application.
Summary of the LOA designer functionalities

We built the LOA graphical editor with ObeoDesigner using the following elements of design:

– Integration of the four views into a single editor with Layers
  – Nodes and Containers are used to create ECore-like class diagrams
    » Child nodes (i.e. attributes) and Border nodes (i.e. similar to components ports) have conditional styles
  – Element-based relationships are used to draw mappings between nodes *
  – Descriptions (Double Click * and Drop *) capture interactions from users and perform changes

– Single viewpoint with single diagram (multi-views on a single representation instead of multiple model editing tasks)

Design of the main elements of the diagram was almost smooth.
Getting around ObeoDesigner limitations and/or lacks in the description model

Behavior

• Descriptions
  – DoubleClick descriptions are used to display a code editor popup
    » Difficult to reuse in multiple places
  – Drop descriptions allow moving an instance of a class to another class
    » Not able to decipher how to properly move child bordered nodes between containers
    » Use of an External Java Actions instead

Structure

• Operations
  – Collections are not supported in the current set of operations provided in the editor
    » Basic operations such as add or remove are written in Java

• Hierarchy
  – Accessing data in superclasses
    » Accessing attributes or references from any level of the hierarchy of the current object is written in Java

• Edges
  – Proper creation and display of a mapping requires one node and two edges + external java actions
    » Element-based edges do not support complex decorations to display a shape between two “lines”
    » Make the deletion of a mapping complex because the components of the mapping are not “bound”

External Actions

• Single statement method
  » Code Editor Popups: Need an EJA for executing a single statement

• Working on edges
  » Requires the use of internal classes Oo10226oo & Oo10250oo to access data to remove

Importing Existing Design

• Rewriting a class diagram editor from scratch is time-consuming
  » Reusing existing editors such as Ecore diagram editor would have been nice
Major limitations

- Filters are not easy to configure
  - Current implementation removes “views” entirely, fine-grained filtering would have been better
  - Hiding the visual representation of a container while displaying its containees is not supported
  - Difference between Filters and Concerns is unclear

- Visual Elements
  - Bordered nodes are not as configurable as nodes
    - Size of the shape is changeable but does not apply
    - Bug Report has been submitted

- Problem Markers
  - Reporting errors from an external validation step has not been possible; we could not succeed in identifying the proper OD objects to feed with problems data.

Comparison with GMF

<table>
<thead>
<tr>
<th>Effort</th>
<th>Global Complexity</th>
<th>Simple design</th>
<th>Complex Design</th>
<th>Shapes customization</th>
<th>Reuse external design</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMF</td>
<td>High</td>
<td>High</td>
<td>High + programming</td>
<td>Identical</td>
<td>Medium</td>
</tr>
<tr>
<td>Obeo Designer</td>
<td>Medium</td>
<td>Low</td>
<td>Medium + EJA</td>
<td></td>
<td>High</td>
</tr>
</tbody>
</table>